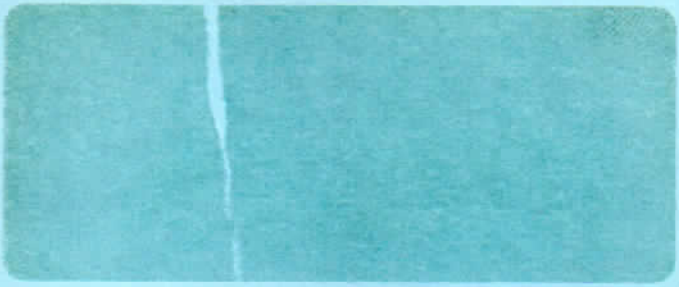

The COMPLETE reference to all the useful routines and facilities of the Spectrum ROM.

With addresses, and information on their use.

!!! DERBAY Ltd

THE GUIDE TO THE SPECTRUM ROM

by Andrew Pennell

Hilderbay Ltd
8/10 Parkway
Regents Park
London NW1 7AA
UK

Tel 01-485-1059 Tlx 22870

THE

GUIDE TO THE

SPECTRUM ROM

by Andrew Pennell

THE GUIDE TO THE SPECTRUM ROM

by Andrew Pennell

CONTENTS

APPENDICES

THE GUIDE TO THE SPECTRUM ROM

by Andrew Pennell

CONTENTS

INTRODUCTION

This book is intended for anyone who wishes to make the most of their Spectrum, using its ROM routines. It assumes some prior knowledge of Z80 assembly language and Spectrum BASIC.

It contains a list of useful subroutines, and describes clearly what they do and examples of how to use them. In addition, details are given of useful Input/Output ports, system variables, and a full explanation of the floating point routines. Numbers preceded with a '$' sign are in hexadecimal.

It is hoped that a supplement will be produced when the Microdrives and RS232 board are released, as their routines are not in the current ROM.

CHAPTER 1
WHERE TO PUT MACHINE-CODE PROGRAMS

Very small routines can be put in the User-defined graphics area, but the best place is above RAMTOP, as mentioned in chapter 26 of the Sinclair manual. There follows a listing of a hex loader, to allow the easy entry of any machine-code programs, including the ones in this book. The hex data can be included in DATA statements from line 100 onwards.

```
10 CLEAR 32499:REM 65267 for 48K
20 LET P=32500:REM 65268 for 48K
30 LET A$=""
40 DEF FN A(A$)=CODE A$-48-7*(A$>"9")
50 IF A$="" THEN READ A$:IF A$=" STOP " THEN
STOP
60 LET B$=A$( TO 2):LET B=16*FN A(B$(1))+FN
A(B$(2))
70 PRINT B$;"  ";
80 POKE P,B:LET P=P+1
90 LET A$=A$(3 TO ):GO TO 50
99 REM Example hex program
100 DATA "016300C9"
999 DATA " STOP "
```

This program allows up to 100 bytes of code, and to execute it use a function such as PRINT USR 32500 (or 65268 for 48K)
The sample data is the routine given in the manual.

(Make sure that you are in upper-case mode when entering the hex data).

In your own routines, avoid the use of registers I, IY and HL'. Changing I will confuse the ULA, and may result in a picture break-up. Index register IY contains $5C3A, and can be used to access the system variables, but never change its value while interrupts are enabled or a crash will occur. Register pair

HL' should not be changed as it is used to hold return addresses used by the floating-point routines in BASIC. When a USR routine is jumped to, BC contains the location of the routine and HL always contains $2D28, which is where the routine RETurns to. The value passed to BASIC is the value of the BC register.

The following user-defined functions can be used to easily convert between decimal and hex:

```
10 DEF FN A$(A)=CHR$(A+48+7*(A>9))
20 DEF FN B$(A)=FN A$(INT(A/16))+FN
A$(A-16*INT(A/16))
30 DEF FN C$(A)=FN B$(INT(A/256))+FN
B$(A-256*INT(A/256))
40 DEF FN A(A$)=CODE A$-48-7*(A$>"9")
50 DEF FN B(A$)=16*FN A(A$(1))+FN A(A$(2))
60 DEF FN C(A$)=256*FN B(A$( TO 2))+FN B(A$(3
TO 4))
```

FN B$ and C$ convert a decimal number to a hexadecimal string of one, or two bytes respectively, and FN B and C perform the reverse i.e. convert a hex string into a decimal number.

CHANNELS

For all input and output (except by cassette), channels are used. These are numbered from -3 ($FD) to 15 ($0F). Channels $FD to $FF are used by the system, and best avoided, but channels 0 to 3 can be used to good effect. Channels 0 and 1 are identical, and correspond to the two lower lines for output, and the keyboard for input. Channels 2 and 3 are output only, and correspond to the upper screen, and ZX printer, respectively. Channels 4 to 15 are not yet used, though some will be used for the Microdrives, RS232 and Networking. It is also possible, though not easy, to use ones own routines, and I have written software, using channel 15, to print 42 characters per line.

OPENCHAN $1601

This opens the channel number in register A for input and output. No checking is done on the value of A, so ensure it is between $FD and $0F. If no channel exists, error O will result.

OUTPUT $0010

This outputs the character in the A register to the currently selected channel, including control characters. It can be used with the one-byte RST 10 instruction.

INPUT $15D4

This loads the A register with the next character from the current channel. Channels 0 and 1 can be used to read the keyboard in the normal way, but attempting an input from channels 2 or 3 will result in error J.

PRINTSTRING $203C

This outputs a string of BC bytes stored in

memory from (DE) onwards onto the current channel.

PRINT $0C0A 3092

This routine prints a message on the current channel from a list stored in memory. Up to 255 messages may be stored in the list sequentially, with the last character in each message having bit 7 set. There are several such lists in the ROM, and the error list can be shown with the following BASIC program:

```
10 FOR I=5010 TO 5460
20 LET A=PEEK I:IF A>128 THEN LET A=A-128
30 PRINT INVERSE(PEEK I>128);CHR$(A);
40 NEXT I
```

To use the routine, DE must point to the start of the list in memory, and A must hold the number of the required message. Expanded characters (codes $A5 to $FF) can be output by loading A with the code required less $A5, and executing a CALL $0C10. The keyword list itself starts at $0096, and the cassette tape messages begin at $0A92.

PRINTBC $1A1B 6683

This prints the value of BC in decimal on the current channel, up to a limit of 9999. It is used by the BASIC to print line numbers (although the entry point is different).

Example:To print the words 'Nonsense in BASIC' in the centre of the screen.

```
3E 02      LD A,02          ;select upper screen
CD 01 16   CALL OPENCHAN
3E 16      LD A,22          ;AT control code
D7         RST 10           ;output it
3E 0B      LD A,11          ;Y position
D7         RST 10           ;output it
```

```
3E 07       LD A,07          ;X position
D7          RST 10           ;output it
3E 0B       LD A,11          ;11th message
11 92 13    LD DE,$1392      ;start of error list
CD 0A 0C    CALL PRINT       ;output it
C9          RET              ;back to BASIC
```

CHAPTER 3 - THE SCREEN

The screen memory is divided into two parts - the first is the high-resolution bit-map, and lies between 16384 and 22527. Each pixel on the screen is represented by one bit in memory, although it is not mapped in a very straight-forward way. For exact details of how it is mapped see page 164 of the Sinclair manual, though it can be demonstrated with the following BASIC program:

```
10 FOR I=16384 TO 22527
20 POKE I,255
30 NEXT I
```

The second part is the colour attribute map, and consists of 768 bytes starting at 22528. Each byte controls the colour of a character square, of 8 pixels in size, i.e. 8 bytes in the bit-map. It is arranged in the expected order e.g. byte 22528 corresponds to the top left character square. Each bit controls a certain colour function: Bits 0-2 are the INK colour, bits 3-5 determine the PAPER colour, bit 6 selects BRIGHT, and bit 7 selects FLASH. Permanent colour attributes can be set from machine-code by altering the variables ATTR P and MASK P, and temporary colours by altering ATTR T and MASK T (see appendix E for details). Temporary attributes can also be set by outputting the appropriate control codes on to channel 2.

Fortunately there are several routines that can be used to simplify programs.

CHUNKY $0B38 2872

When using your own routines to print characters on the screen, it is useful to know where the bit-maps for the character set are. The layouts of ASCII codes 32 to 127 are stored

according to the CHARS system variable ($5C36), and codes 144 to 164 are stored according to the UDG variable, location $5C7B. You may wonder where the 'chunky' graphics, codes 128 to 143 are stored. The answer is simple - they aren't! When it is required to put them on the screen, the ROM calculates the eight bytes and stores them in locations $5C92 to $5C9A, which is actually the MEMBOT area, but never used at this time. To use the routine, the A and B registers should each be loaded with the required character code. Upon return from the routine, the bit pattern for the character is in the first eight bytes of MEMBOT.

CALCULATE $0E9E 3742

This routine works out the location of the first byte in a particular line on the screen. The A register must contain the line number (with 0 corrsponding to the top line, and 23 the very bottom), and the routine will return the location in the HL register pair.

COLOUR $0BDB 3035

This routine fills the colour byte corresponding with the screen bit location HL with the current colour attributes.

PLOT $22DF 8927

This routine is directly equivalent to the PLOT command in BASIC. The C register should contain the X co-ord, and B the Y co-ord. Permanent and temporary attributes are taken into account.

BITCALC $22AA 8874

This calculates which bit of which screen location corrsponds to the pixel (C,B). On return, the HL register contains the location, and the A register the bit of that location to be set. Thus, a simple UNPLOT routine would be:

```
0E xx       LD C,x co-ord
06 yy       LD B,y co-ord
CD AA 22    CALL BITCALC
47          LD B,A
04          INC B
3E FE       LD A,FE     ;BIN 11111110
0F       L1: RRCA        ;rotate bit
10 FD       DJNZ,L1     ;into correct place
A6          AND (HL)    ;ignore bit
77          LD (HL),A   ;put it in screen
C9          RET         ;back to BASIC
```

It can also be used for true 256x192 resolution, with the origin at the top left by loading A with the Y co-ord, C with the X co-ord, and calling $22B0.

CLS $0D6B

This is directly equivalent to the CLS command in BASIC, and clears the whole screen, filling the attribute file with the current permanent colours. This routine changes the current channel, so if it is required to preserve the channel, use the following instructions:

```
2A 51 5C    LD HL,(CHURCHL)
E5          PUSH HL
CD 6B 0D    CALL CLS
E1          POP HL
22 51 5C    LD (CHURCHL),HL
```

SCROLL $0DFE

This routine scrolls the whole screen up by one character line, including the colour attributes. It does not effect the SCR CT system variable.

SCROLLsome $0E00

This routine scrolls B character lines from the bottom upwards, including the attribute file.

DRAWstraight $2477

This is identical to the equivalent BASIC statement. Before calling it, the top item on the calculator stack should be the Y displacement, and the next item the X displacement. For an explanation of the calculator stack, please see chapter 8.

DRAWcurve $2394

This is identical to the BASIC statement with three arguments. Before using it, the top item on the calculator stack should be the angle, the next item the Y diplacement, and the next the X. See appendix A for details of a bug in this routine.

CIRCLE $232D

In a similar way to DRAWcurve, the radius, Y co-ord and X co-ord should be the top items on the calculator stack, with the radius uppermost.

IMPORTANT:Please note that the value of register HL' is altered by the above three routines, so the user must preserve its value e.g.

```
D9          EXX
E5          PUSH HL
D9          EXX
CD 2D 23    CALL CIRCLE
D9          EXX
E1          POP HL
D9          EXX
```

Providing interrupts are enabled, the keyboard is scanned every 50th of a second, to allow the system to detect how long a key is being pressed for.

One way to read the keyboard is to use the input facility of channel O. Before calling INPUT to read it, bit 3 of FLAGS should be reset, and bit 5 set. This inhibits the printing of the current line on the screen. On return from INPUT, the accumulator contains the code of the key pressed. Unfortunately, when CAPS LOCK or GRAPHICS keys are pressed, the current line appears on the screen for a while. (This is the cause of the SCROLL? bug, see appendix A).

Another way to scan the keyboard is to test bit 5 of FLAGS - if it is zero, then no key has been pressed. If it is not, then the ASCII value of the key can be read from LASTK, at location $5C08. Bit 5 of FLAGS should then be reset, to inform the monitor that the key has been noticed. The contents of LASTK depend on the current cursor mode, which is stored in $5C41. For L or C mode it is zero, for E mode it is 1, for G mode it is 2, and for K mode it is 4.

Alternatively, location $5C03 always contains the code of the key currently being held down, or $FF if none are. It ignores the cursor mode, and always holds the upper-case ASCII of the key. Bit 5 of FLAGS is not required to use it.

The actual keyscan routine is at $0038, the maskable interrupt location (in interrupt mode 1). The only times that interrupts are disabled by the ROM is for time-critical operations, namely printer, sound and tape operations, and probably the Microdrives.

It is possible to 'lock' a BASIC program so that once it has started, it is impossible for

it to stop, and for it to be copied. This can be done by an auto-run program, which immediately executes a small routine to disable the interrupts. Of course, INKEY$, PAUSE and INPUT statements will not work, and printer, tape and sound commands should be avoided as they will re-enable interrupts. This idea also has the advantage that the BASIC program will speed up by about 10%. If at any time, BREAK is attempted, the system will 'hang-up', as a HALT instruction is encountered when any error occurs, or when a program finishes.

If multiple key depressions are required (e.g. in 2-player games) the IN instruction should be used, as explained (for the similar BASIC statement) on page 160 of the Sinclair manual.

BREAK $1F54

This routine does not require interrupts to be enabled, and scans the keyboard to detect BREAK. If it is being pressed, the carry flag is reset, else it is set.

To output characters to the printer, channel 3 should be selected, as mentioned in chapter 2. However, if you wish to write your own printer routines, there are several useful ROM subroutines.

The printer buffer consists of 256 bytes, starting at location $5B00. This is mapped in a straight-forward manner, with each sequence of thirty-two bytes representing 256 dots in a printed line. There are eight such bit lines to a full character line.

The system variable at location $5C81, labelled 'not used' in the manual, actually always contains $5B, as it is the most significant byte of the next LPRINT position, and is used in conjunction with the PR CC variable at location $5C80.

EMPTY $0EDF 3807

This routine clears the printer buffer and resets its system variables.

LLINE $0EF4 3828

This copies the bit patterns of the 32 bytes from (HL) onwards onto the printer. Interrupts must be disabled before the routine is used, and when all printer output is finished the following instructions must be used:

```
3E          LD A,04
D3 FB       OUT (FB),A  ;stop motor
FB          EI          ;enable interrupts
```

LPRINT $0ECD 3789

This transfers the contents of the printer buffer to the printer, empties the buffer and resets its variables. All handling of interrupts and stopping the printer is handled by the routine.

COPY $0EAC 3756

This is the direct equivalent to the BASIC command, and copies the upper 22 lines to the printer. If it is desired to copy all 24 lines, use the following routine:

```
F3          DI          ;disable interrupts
06 C0       LD B,192    ;192 lines
C3 AF 0E    JP $0EAF
```

CONTROL PORT $FB

The printer hardware is controlled by the I/O port $FB. When writing to the port, setting bit 1 will slow the printer motor, setting bit 2 will stop it altogether, and setting bit 7 applies a high voltage to the stylus. When read, bit 0 is the signal from the syncronising disc, bit 6 is low when the printer is connected, and bit 7 is high when the stylus is on the paper.

CHAPTER 6 - THE CASSETTE ROUTINES

Programs on cassette consist of two sections, each preceeded with a constant tone to prevent problems with the ALC circuitry of some tape recorders. Between the two sections there is a pause of exactly one second, but this is only a minimum with regards to LOADing a program.

All the tape routines have the same entry point, namely $075A. The function executed depends on the contents of the least-significant byte of the T ADDR variable at location $5C74, which correspond with the following BASIC functions:

0 SAVE
1 LOAD
2 VERIFY
3 MERGE

To use these facilities from a machine-code program, a data block of 17 (for SAVE) or 34 (for any other function) bytes must be set up, starting at location (IX). The ROM uses the temporary workspace area, but there is no actual restiction. The byte at (IX) should contain a number referring to the file type:

0 BASIC program
1 Character array
2 Number array
3 Bytes (or SCREEN$)

The bytes (IX+1) to (IX+$0A) should contain the filename, consisting of normal ASCII codes only. The null string filename (used for LOAD, VERIFY and MERGE) is represented by (IX+1) containing $FF. The bytes (IX+$0B) to (IX+$10) contain various parameters, depending on the type of file. For functions other than SAVE, allowance should be made for the extra 17 bytes, that are used for reading the tape. All two-byte numbers should be stored in the usual

Z80 reverse way, and any unused bytes in the data block should contain spaces - code 32.

BASIC PROGRAM

(IX+$0B),(IX+$0C) should contain the length of the program and variables i.e. ELINE-PROG-1.
(IX+$0D),(IX+$0E) should contain the line number execution will jump to, or $8020 if auto-execution is not required.
(IX+$0F),(IX+$10) should contain the length of the variables area i.e. ELINE-VARS.

DATA

(IX+$0B),(IX+$0C) should contain the length of the whole array in memory. (IX+$0E) should contain the byte holding the name and type of variable i.e. the leftmost byte shown on pages 166-168 of the manual. See appendix A for details of a bug in this routine.

BYTES (AND SCREEN$)

(IX+$0B),(IX+$0C) should contain the number of bytes to be saved. (IX+$0D),(IX+$0E) should contain the location of the first byte to be saved or loaded.

When T ADDR and the data block have been set up as described, HL should be loaded with the address of the first byte to be handled, and IX should point to the start of the data block. The routine at $075A can then be called.

There are also two routines that are more general in use:

SAVE $04C2

This outputs DE bytes onto tape, starting with location (IX). If the standard format is used, the accumulator should be zeroed for the first section of data, then contain $FF for the second. To generate a pause of one second, use the following routine:

```
06 32       LD B,50
76      L1:HALT
10 FD       DJNZ,L1  ;wait for fifty 50ths of a
second
```

LOAD $0556

This loads DE bytes from tape, and stores them from (IX) onwards. As with the SAVE routine, the accumulator should contain zero for the first section, and $FF for the second, with the carry flag set in both cases.

AUTOMATIC EXECUTION

At first sight, it seems impossible to have an auto-run machine-code program on cassette. The method on page 181 of the manual works, but is clumsy. The way round this is to save the BASIC program with the machine code, in one file. This can be done by saving the whole of locations 23552 to the end of RAM. For example, if you have a 16K Spectrum, and a machine-code routine above RAMTOP starting at 30000, you could use the following one-line BASIC program to save it so that it will auto-execute:

```
10 SAVE "FILENAME"CODE
23552,32768-23552:RANDOMIZE USR 30000
```

The SAVE must be done from within a program, else the system will crash. To load (and execute) this program, use

```
LOAD "FILENAME"CODE
```

Please note that verification of such files is impossible.

There are fifty commands in Spectrum BASIC, some of which are very useful in machine-code routines. For reference, appendix C lists all of them, and their locations. It is interesting to note that all the Microdrive commands jump to location $1793, and produce an 'Invalid Stream' error message.

CLEAR $1EAF 7855

This is directly equivalent to the BASIC statement CLEAR BC. It moves RAMTOP to location BC, clears all variables,and does RESTORE,CLS and PLOT 0,0 statements. If BC is zero then RAMTOP is left unchanged.

PAUSE $1F3D 7997

This will wait for BC 50ths of a second, or until a key is pressed. If BC is zero, it will wait indefinitely. As there is a HALT instruction in this routine, interrupts should be enabled beforehand.

BORDER $2297 8855

This sets the screen border colour to the value of the A register, and updates BORDCR.

BEEP $03B5 949

This routine produces an audible tone dependent on HL, for a period dependent on DE. Alternatively, it is possible to follow the exact BASIC syntax of BEEP x,y by putting x then y on the calculator stack and executing a CALL $03F8.

ERROR $0008

There are 28 different error messages in BASIC, and these can be a helpful way to terminate a machine-code routine. The routine at $0008 can be easiest used with a RST 08

instruction. The byte following it is taken as one less than the required report code, and, when executed, will return control to BASIC with an appropriate message. There are two other ways to produce error messages: The first is to load the L register with one less than the required report code,and execute a JP $0055 instruction. The other is most useful for conditional errors, for example if the carry flag is set by a computation. This can be done by executing a conditional jump to a certain location in memory. A list of locations for this purpose is given in appendix D.

CHAR $0018

This loads the accumulator with the character in the BASIC program currently being interpreted, ignoring any colour control code. A one-byte RST 18 instruction can be used.

NXTCHAR $0020

This loads the accumulator with the next character of the program, ignoring colour control codes. A one-bye RST 20 instruction can be used.

FLOATPT $0028

The RST 28 instruction is used for floating point operations. It is explained fully in a later chapter.

MAKEROOM $0030

This increases the workspace area by BC bytes, and moves the calculator stack up accordingly. On return, DE contains the address of the first free byte.

INTERRUPT $0038

This is normally executed automatically every 50th of a second. It increments the FRAMES counter, and scans the keyboard.

NMASKINT $0066

This is the non-maskable interrupt routine and is potentially very powerful. At present, no INT signals are received by the Z80, so must be generated externally (e.g. by the Expansion Module). As it stands, the routine, if executed, results in the machine resetting unless either of $5CB0 or $5CB1 contain non-zero numbers. It appears that it has been designed for any interrupting hardware to set bit 4 of location $006D (converting a JR NZ to JR Z). If this is done, then the routine pointed to by $5CB0/1 is executed when a NM interrupt is received. This facility seems ideal for RS232 input.

GOTO

It is sometimes desirable to return to BASIC at a particular section of program other than that section which contains the USR function. This can be accomplished by loading NEWPPC ($5C44) with the line number, and NSPPC ($5C44) with the statement number, before the RETurn to BASIC.

I/O PORT $FE

The I/O port $FE is used to control several functions. When reading from it, bits 0 to 4 scan the keyboard, in conjunction with the eight upper bits of the address lines (see page 160 in the manual), and bit 6 is the value of the signal at the EAR socket. When writing to it, bits 0 to 2 determine the BORDER colour, bit 3 drives the MIC socket, and bit 4 drives the loudspeaker.

CHAPTER 8
FLOATING POINT ROUTINES

The floating-point mathematics routines account for a large amount of the Spectrum ROM. They are obviously useful in machine-code routines that require complex calculations, but is sometimes necessary to use them for other ROM routines, such as CIRCLE.

There are two main areas in memory used to manipulate floating point numbers - the calculator stack and MEMBOT.

THE CALCULATOR STACK

This is a section of memory used for storing numbers and strings in groups of five bytes. It is the correct way up (unlike the machine stack) and has two system variables associated with it - STKBOT (at $5C63) and STKEND (at $5C65) which store the bottom and top of the calculator stack, respectively.

MEMBOT

This consists of thirty bytes that are used to store six numbers, from locations $5C92 to $5CAF. The variable MEM (at $5C68) usually contains $5C92, but this changes when the interpreter handles FOR-NEXT variables. The reason for this may be apparent when one sees the nature of these variables, on page 167 of the manual.

NUMBERS

As has been mentioned, numbers are stored in five bytes. This is done in one of two ways - for integers between -65535 and 65535:

(i) the first byte is zero
(ii) the second byte is zero for a positive number, or $FF for a negative number
(iii) the third and fourth bytes are the less and more significant bytes of the number (or

the number+65536 if it is negative)
(iv) the fifth byte is zero.

All other numbers are stored in the form explained on page 169 of the manual. Note that there are bugs in some floating-point routines in the ROM that arise because the number -65536 can be stored in both forms. See appendix A for details.

STRINGS

Strings can also be represented by five bytes:
(i) the first byte is zero
(ii) the second and third bytes are the location of the start of the string, in the usual format
(iii) the fourth and fifth bytes are the length of the string.

In addition to this, bit 5 of FLAGS is reset when a string has been placed on the top of the calculator stack. The strings themselves are stored in the variables area and the temporary workspace.

There are several useful routines for handling registers and the calculator stack, hence referred to just as the stack.

STACKA $2D28 11560
This places the value of the accumulator on the stack.

STACKBC $2D2B 11563
This places the value of the register pair BC onto the stack.

STACKDE $2CB3 11443
This places the value of the register pair DE onto the stack.

UNSTACKA $1E94 7828

This removes a number from the top of the stack, rounds it to the nearest whole number (not just INTed) and places it in the accumulator. If it is negative, or greater than 255 then error B results.

UNSTACKBC $1E99 7833

This removes a number from the top of the stack, rounds it to the nearest whole number, and places it in BC. If it is negative, or greater than 65535 then error B results.

PRINTSTACK $2032 8242

This removes the top item from the stack, and prints it on the current channel. It works for both numbers and strings.

STACK $2AB6 10934

This places the values of the registers A,E,D,C,B onto the top of the stack, in that order (register A corresponding to the exponent byte).

STACKSTRING $2AB1 10929

This places a string on the top of the stack. DE should contain the location of the start of the string, and BC its length.

UNSTACK $2BF1 11249

This removes the five bytes from the top of the stack, and places them in registers B,C,D,E,A in that order. For a string, BC will contain its length, DE its start, and A will be zero.

STACKVARS $35BF 13759

This loads DE with STKEND, and HL with STKEND-5.

RST 28 INSTRUCTION

To perform operations on the numbers and strings on the calculator stack, the one-byte RST 28 instruction is used. After this $EF byte, a sequence of data bytes should follow, ending with the EXIT byte $38. The list below the actions of all of them, and dollar signs are used to discriminate numeric arguments from strings. The contents of the top of the stack after each operation are also shown.
Before-hand, the argument A is assumed to be on the top of the stack, with B below it. An asterisk denotes that the result is determined by other factors, explained more fully below.

RST 28 BYTE TABLE

BYTE	NAME	STACK RESULT	
(Initially		B	A)
00	JRNZ	B	
01	SWOP	A	B
02	REMOVE	B	
03	SUBTRACT	A-B	
04	MULTIPLY	A*B	
05	DIVIDE	A/B	
06	POWER	A^B	
07	OR	A OR B	
08	AND	A AND B	
09	<=	A<=B	
0A	>=	A>=B	
0B	<>	A<>B	
0C	>	A>B	
0D	<	A<B	
0E	EQUAL	A=B	
0F	ADD	A+B	
10	AND $	A$ AND B	
11	<=$	A$<=B$	
12	>=$	A$>=B$	
13	<>$	A$<>B$	

20 14	>$	13627	A$>B$		
21 15	<$	13627	A$<B$		
22 16	EQUAL$	13627	A$=B$		
23 17	ADD$	13724	A$+B$		
24 18	VAL$	13790	B		VAL$ A$
25 19	USR$	13500	B		USR A$
26 1A	CHANNEL	13893	B		A *$
27 1B	NEGATE	13422	B		-A
28 1C	CODE	13929	B		CODE A$
29 1D	VAL	13790	B		VAL A$
30 1E	LEN	13940	B		LEN A$
31 1F	SIN	14261	B		SIN A
32 20	COS	14250	B		COS A
33 21	TAN	14298	B		TAN A
34 22	ARCSIN	14387	B		ARCSIN A
35 23	ARCCOS	14403	B		ARCCOS A
36 24	ARCTAN	14306	B		ARCTAN A
37 25	LN	14099	B		LN A
38 26	EXP	14020	B		EXP A
39 27	INT	13999	B		INT A
40 28	SQR	14410	B		SQR A
41 29	SGN	13458	B		SGN A
42 2A	ABS	13418	B		ABS A
43 2B	PEEK	13484	B		PEEK A
44 2C	IN	13477	B		IN A
45 2D	USR	13491	B		USR A
46 2E	STR$	13855	B		STR$ A
47 2F	CHR$	13769	B		CHR$ A
48 30	NOT	13569	B		NOT A
49 31	COPY	13248	B		A A
50 32	MOD	13984	B	MOD A	INT(B/A)
51 33	JR	13958	B		A
52 34	STACKDATA	13254	B		A *
53 35	DJNZ	13946	B		A
54 36	NEGATIVE?	13574	B		A<0
55 37	POSITIVE?	13561	B		A>0
56 38	EXIT	13979	B		A
57 39	CHEBYSHEV	14211	B		F(A)
58 3A	COMPACT	12820	B		A
59 3B	B-FUNCTION	13218	B		A *
60 3C	DECIMAL	11599	B		*

3D	EXPAND		B	A		
A0	STACKZERO		B	A	0	
A1	STACKONE	"	B	A	1	
A2	STACKHALF	"	B	A	0	0.5
A3	STACKPI/2	"	B	A	PI/2	
A4	STACKTEN	"	B	A	10	
C0	STORE0		B	A		
C1	STORE1	"	B	A		
C2	STORE2	"	B	A		
C3	STORE3	"	B	A		
C4	STORE4	"	B	A		
C5	STORE5	"	B	A		
E0	RECALL0		B	A	*	
E1	RECALL1	"	B	A	*	
E2	RECALL2	"	B	A	*	
E3	RECALL3	"	B	A	*	
E4	RECALL4	"	B	A	*	
E5	RECALL5	"	B	A	*	

Most of the bytes perform normal mathematical functions upon the calculator stack, but some are not so quite straight-forward.

JRNZ Byte 00

This byte must be followed with a jump byte. This function removes the top item on the stack, and if it is zero then the following byte is skipped over. If it is not, a relative jump is performed to another RST 28 data byte. This jump can be forwards or backwards, and is determined by the byte following the $00. This is calculated in a similar way to noraml Z80 JR bytes, but not exactly. This is best illustrated with an example.
e.g. to cause a program to stop if the stack is not zero:

```
   EF        RST 28
   00        DEFB 'JRNZ'
  ┌►03       DEFB 03       ;this is the jump byte
  │ 38       DEFB 'EXIT'   ;end the f.p. if the
stack is zero
  │ C9       RET
  └►38       Defb 'EXIT'   ;if it isn't zero
   C3 3F 05  JP $053F      ; "invalid argument"
```

This method of using a jump byte is also used with codes $33 (JR) and $35 (DJNZ). If you wish to test for the stack being zero, preceed the JRNZ byte with the NOT byte $30. The jump will then execute if the stack was zero.

CHANNEL Byte $1A

This removes the top number from the stack, and uses it as a channel number. If it is greater then 15 or negative then error B will occur. The channel is opened, and read for a character. This character is then stacked as a string. This does not work for existing channels, so must be intended for the Microdrives and/or RS232.

STR$ Byte $2E

See appendix A for details of the bug in this routine.

JR Byte $33

This executes an unconditional jump to another RST 28 data byte, and must be followed with an appropriate jump byte.

STACKDATA Byte $34

This is used to put certain constants on the calculator stack. The $34 byte must be followed with a number of other bytes that define the number. The first following byte is divided by $40 and the exponent of the number will be either

(i) the remainder+$50 if there is a remainder
or
(ii) the second byte+$50 if there is no
remainder.

The quotient of the division will be one less than the number of further bytes. Any unassigned bytes will be zeroed.
For example, PI/2 is stored as

F1 49 0F DA A2

The first byte is divided by $50 and gives 3 remainder $31. Thus the exponent byte is $50+$31=$81 and there are four more data bytes after $F1. Thus the normal form is:

81 49 0F DA A2

DJNZ Byte $35

The function decrements the contents of BREG ($5C67) and does a relative jump to another RST 28 data byte if it is not zero. The value of BREG is set by the value of the B register just before the RST 28 instruction.

CHEBYSHEV Byte $39

This replaces the top of the stack with a number calculated from a Chebyshev polynomial. This is a formula used as a basis for other maths functions e.g. SIN.

COMPACT Byte $3A

This reduces the floating-point form of the number on the stack to its condensed form, if possible . It has no effect if the number is already condensed.

B-FUNCTION Byte $3B

This takes the value of B REG ($5C67) and uses it as a RST 28 data byte. B REG is determined by the value of B on entry to the RST 28 routine.

DECIMAL Byte $3C

This multiplies the top of the stack by 10^N where N is the value of the Bregister before the RST 28 instruction.

EXPAND Byte $3D

This function is the opposite of the COMPACT function. It examines the number on the top of the stack, and if it is in the compressed form, will expand it to full floating-point form.

STACKHALF Byte $A2

Note that there is a bug in this function such that a spurious zero is stacked beneath 0.5. This can be removed with data byes 01 then 02 (i.e. swop then remove). This is the cause of the DRAW bug (see appendix A).

STOREn Bytes $C0 to $C5

This uses the six sections in MEM of five bytes each to store numbers from the stack. The nibble of the code defines which section the top of the stack is copied into. Unlike the ZX81 equivalent these codes do not remove the top item from the stack.

RECALLn Bytes $E0 to $E5

These are the opposite of the STOREn bytes, and copy a number from a section of MEM (defined by the nibble) onto the top of the stack.

OTHER FUNCTIONS

There are other functions available that calculate then place an item on the stack, that have no RST 28 code.

SCREEN$ $2538 9528

This determines the single character at screen position (B,C), inverted or normal, and places it on the stack. It only recognises normal characters in the ASCII range 32 to 127. If the character is not recognised, a null string results. The bug described in appendix A does not apply when using this routine.

ATTR $2583 9603

This places the colour attribute of screen position (B,C) onto the calculator stack.

POINT $22CE 8910

This examines the pixel (C,B) to see if it is set (i.e. INK colour). If it is, then a 1 is stacked, else a zero is stacked.

PI

The ROM routine cannot easily be used to place PI on the stack, though its method can be copied thus:

```
EF          RST 28
A3          DEFB STACKPI/2    ;stack PI/2
38          DEFB EXIT
34          INC (HL)          ;double it by
                              increasing exponent
```

RND

The easiest way to place RND on the stack is to use the function VAL "RND" thus:

```
3E A5       LD A,'RND'
CD 28 2D    CALL STACKA
EF          RST 28
2F          DEFB CHR$
1D          DEFB VAL
38          DEFB EXIT
```

PASSING VARIABLES BETWEEN BASIC AND MACHINE-CODE ROUTINES

Another use of the floating-point routines is to pass variables between a BASIC program and a machine-code routine. When the interpreter reaches a line such as LET Z=X*USR mcode it will place the value of X on the calculator stack, then execute the routine at location 'mcode'. The routine can then use the value of X either as an integer (e.g. by calling UNSTACKBC) or in its floating point form. It is possible to extend this method to pass a number of variables to the stack e.g.

LET Z=Y+X*USR mcode

will put the values of Y, then X onto the stack. The routine should first check that there are the correct number of variables on the stack, before attempting any calculations. Before the RETurn to BASIC, and appropriate number of variables should be stacked to replace those removed at the beginning, else the system may crash. The easiest way is with a sequence of STACKZERO bytes.

Normally the value returned to BASIC is the contents of the BC register, but this too can be changed to allow decimals to be returned. Normally the return address is $2D2B (the STACKBC routine) but to return to BASIC with the current value of the stack simply remove the $2D2B address from the machine stack, then RET e.g.

```
E1          POP HL
C9          RET
```

An example showing both of these ideas now follows, and gives the Spectrum the MOD function found on more expensive machines, usable from BASIC. It requires a line such as

LET M=B+A*USR MOD

and M will then equal B MOD A.

```
2A 65 5C    LD HL,(STKEND) ;check for at least
five bytes
ED 5B 63 5C LD DE,(STKBOT) ;on calculator stack
A7          AND A          ;reset carry
ED 52       SBC HL,DE
7D          LD A,L
FE 0A       CP $0A
DA 8B 28    JP C,$288B     ;produce error
            ;message if less than 10 bytes
                           ;Stack status
EF          RST 28         ;  B     A
32          DEFB MOD       ;B MOD A INT(A/B)
02          DEFB REMOVE    ;B MOD A
C0          DEFB STOREO    ;B MOD A
02          DEFB REMOVE
A0          DEFB STACKZERO; 0
A1          DEFB STACK1    ; 0       1
E0          DEFB RECALLO   ; 0       1   B MOD A
38          DEFB EXIT
E1          POP HL       ;remove return address
C9          RET          ;back to BASIC
```

The top of the calculator stack is shown at each stage to clarify the procedure. Note the way the original values of A and B are replaced with one and zero, respectively, so the interpreter evaluates M as 0+1*(B MOD A).

The Microdrives and RS232 board will use currently unassigned channels for input and output.

The LOAD,SAVE,VERIFY and MERGE functions will work in a similar way to the tape routines, but on the Microdrives. To open a file, the BASIC command OPEN#A,A$ will be used. The ROM will search for the file named A$ on the drive selected by A, and prepare it to be written to and read from. The function INKEY$#A will read one byte from the file, and INPUT#A,b$ statements will enable variables to be read from it. Data will be inserted into the file with PRINT#A statements. It will also be possible to dump a program listing in normal ASCII, into the file with the LIST#A command.

When all writing and reading from the file has been done, the CLOSE#A statement is required, else the file contents may be lost. The CAT command will list all the files on the Microdrives, and FORMAT A will configure a storage medium for use. MOVE A$,B$ will copy files, and ERASE A$ will delete a file.

The Expansion Module will contain all the necessary electronics, and a new ROM. It may be overlaid with the existing one, but will most probably overwrite it completely. (It is interesting to note the the newest Spectrums have their ROMs soldered in). The locations of most routines should remain unchanged, the new routines using the space from $386E to $3CFF, though the OPEN and CLOSE routines will be changed considerably, Currently, the only valid filenames are "K","S", and "P", which is rather restrictive, to say the least! The SAVE, LOAD, VERIFY and MERGE routines will have to be modified slightly.

There are six bytes from $11D4 to $11D9 that

currently contain NOPs. These seem to be the place for two CALL instructions, to check how many Microdrives are connected, and to set up their memory maps. The byte at $1B06 appears to be incorrect, as it contains $0A instead of $06. This is in the syntax table, and changes its argument from a string to a number. The byte at location $006D in the NMASKINT routine will probaly be changed from $20 to $28 to allow RS232 input.

When the Expansion Module is released it is hoped that a supplement will be released descibing its use from machine-code, and the additional ROM routines.

APPENDIX A - ROM BUGS

There are quite a few bugs in the Spectrum ROM, but most are minor. A full list now follows, with the most serious first.

-65536 error

This bug is caused by the fact that this number can be stored in both expanded and compressed forms. It affects the RST 28 INT byte. It is easily demonstrated with:
INT -65536=-1 (instead of -65536) and
INT -65535.5=-1E-38 (instead of -65535).

SAVE DATA error

If a undimensioned string is saved to tape, when it is reloaded the ROM thinks it is a dimensioned string array, and corrupts it, usually by giving it a large number of dimensions.

STR$ error

When the STR$ function acts upon a non-zero number between -1 and 1 it places a spurious zero beneath the string on the calculator stack. This affects the RST 28 STR$ byte, and prevents reliable string concatenation from BASIC and machine-code.

SCREEN$ error

When used from BASIC, the resultant string is stacked twice on the calculator stack. As with STR$, it prevents string concatenation. It does not occur when using machine-code.

DIM error

If an array is dimensioned to zero, an error message is produced, but the array is deleted from memory, which can sometimes be useful. It is also present on the ZX81.

INPUT errors

1.An INPUT statement can be executed in direct mode, which should not be allowed.

2.An INPUT statement does not actually require any variables, e.g.

```
10 INPUT "This is a bug"
20 GO TO 10
```

3. The statement INPUT#2;A$ produces the expected error message, but the cursor is in a strange place.

REM error

When a colon is put in a REM statement, the machine goes into keyword mode.

THEN error

It is not necessary to follow a THEN statement with anything, e.g.

```
10 IF A=B THEN
```

This bug is also present on the ZX81.

CLOSE error

Attempting to CLOSE channels 4 to 15 without first OPENing them results in either a strange error message, or a crash.

PRINT errors

These bugs are not obvious until attempting to use any channel for your own output routines. When doing so, the functions CHR$ and + (for string concatenation) will not work in PRINT statements, and sometimes crash the system.

APPENDIX B - MANUAL MISPRINTS

Unfortunately there are a few misprints in the Sinclair manual, which can cause confusion:

Page 114: In the last paragraph, "9 gives CHR$ 19 and CHR$ 1 for extra brightness" and "CAPS SHIFT with 9 gives CHR$ 18 and CHR$ 1 for flashing"

Page 152: The last line should be "Make a printed graph of SIN by running the program in Chapter 17 and then use COPY".

Page 170: The second last line should be "(or the number+65536 if it is negative)"

Page 173: The first line should start "The bytes in memory from 23552....".

Page 176: Line 20 in the first program should be

```
20 PRINT PEEK( PEEK 23637+256*PEEK 23638+N)
```

and in the second program

```
20 PRINT PEEK( PEEK 23635+256*PEEK 23636+N)
```

The programs given are for the ZX81.

Page 185: CHR$ 92 is "\", not "/"

Page 202: There is no such command as DELETE.

APPENDIX C - BASIC COMMAND ROUTINES

This is a list of the BASIC commands and their ROM routine locations, in hex.

DEF FN	1F60	8032
CAT	1793	6035
FORMAT	1793	6035
MOVE	1793	6035
ERASE	1793	6035
OPEN #	1736	5942
CLOSE #	16E5	5861
MERGE	0605	1541
VERIFY	0605	1541
BEEP	03F8	1016
CIRCLE	2320	8992
INK	1C96	7318
PAPER	1C96	7318
FLASH	1C96	7318
BRIGHT	1C96	7318
INVERSE	1C96	7318
OVER	1C96	7318
OUT	1E7A	7802
LPRINT	1FC9	8137
LLIST	17F5	6133
STOP	1CEE	7406
READ	1DED	7661
DATA	1E27	7719
RESTORE	1E42	7746
NEW	11B7	4535
BORDER	2294	8852
CONTINUE	1E5F	7775
DIM	2C02	11266
REM	1BB2	7090
FOR	1D03	7427
GO TO	1E67	7783
GO SUB	1EED	7917
INPUT	2089	8329
LOAD	0605	1541
LIST	17F9	6137
LET	1544	~~5444~~ 573

PAUSE	1F3A	7994
NEXT	1DAB	7595
POKE	1E80	7808
PRINT	1FCD	8141
PLOT	22DC	8924
RUN	1EA1	7841
SAVE	0605	1541
RANDOMIZE	1E4F	7759
IF	1CF0	7408
CLS	0D6B	3435
DRAW	2382	9090
CLEAR	1EAC	7852
RETURN	1F23	7971
COPY	0EAC	3756

APPENDIX D - ERROR LOCATIONS

This is a list of error messages and the bytes to follow a RST 8 instruction to produce them. Also given are the locations that can be jumped to to produce the required error message. Please note that there is no location to produce a 'No room for line' error. All numbers are in hex.

BYTE	ERROR	LOCATION
FF	OK	1BB0
00	NEXT without FOR	1DD8
01	Variable not found	0670
02	Subscript wrong	2A20
03	Out of memory	1F15
04	Out of screen	0C86
05	Number too big	31AD
06	RETURN without GOSUB	1F38
07	End of file	15E4
08	STOP statement	1CEE
09	Invalid argument	053F
0A	Integer out of range	046C
0B	Nonsense in BASIC	1C8A
0C	BREAK - CONT repeats	0552
0D	Out of DATA	1E08
0E	Invalid filename	0642
0F	No room for line	
10	STOP in INPUT	21D4
11	FOR without NEXT	1D84
12	Invalid I/O device	15C4
13	Invalid colour	2244
14	BREAK into program	1B7B
15	RAMTOP no good	1EDA
16	Statement lost	1BEC
17	Invalid stream	160E
18	FN without DEF	2812
19	Parameter error	288B
1A	Tape loading error	0806

APPENDIX E - SYSTEM VARIABLES

There are 180 bytes in RAM set aside by the operating system to store certain values, and many are useful to the machine-code programmer. A full list of these in chapter 25 of the manual, but it is not made particularly clear what some of them do. There follows a list of the most useful, and their exact function. The index register IY contains $5C3A, and can be used to access all the variables.

$5C03 KSTATE3 stores the upper-case ASCII of the key currently being held down, ignoring any SHIFT keys and MODE, or $FF if no keys are being pressed.

$5C08 LAST K stores the code of the last key to be pressed (see chapter 4).

$5C10 STRMS contains 38 bytes which are used in conjunction with CHANS to store the locations of the input and output routines for each channel. By changing these it is possible to use your own routines.

$5C3B FLAGS If bit 0 is reset, then a space will be printed before the next keyword. Bit 1 is set when the printer is the current channel output device. Bit 5 is used in reading the keyboard (see chapter 4), and bit 6 is set when a number is on the top of the calculator stack, or reset when a string is. Bit 7 is reset when a BASIC line is being checked for syntax, and set when being executed.

$5C3C TVFLAG Bit 0 is reset when the upper screen is the current channel output device.

$5C41 MODE determines the cursor mode, and contains 0 for L or C mode, 1 for E mode, 2 for

G mode, or 4 for K mode.

$5C48 BORDER contains certain colour attributes. Bits 0-2 determine the INK colour of the lower screen, bits 3-5 the PAPER and BORDER colour, bit 6 controls the BRIGHTness, and bit 7 the FLASHing.

$5C4F CHANS stores the start of the data for the input/output routines. With no Microdrives connected this is $5CB6.

$5C68 MEM stores the address of the area used for the calculator memory. This is usually MEMBOT ($5C92).

$5C6A FLAGS2 Bit 1 is reset when the printer buffer is empty. Bit 3 is the case flag, and is set when in upper-case.

$5C6B DFSZ contains the number of lines in the lower part of the screen. The equivalent variable on the ZX81 could be zeroed to allow full screen printing, but this is not recommended on the Spectrum as a crash is very likely.

$5C74 TADDR is useful to determine which cassette operation is required (see chapter 6).

$5C81 'Not used' always contains $5B which is the most-significant byte of the printer cursor position.

$5C84 DF CC contains the address in the screen bit-map of the current cursor position.

$5C88 SPOSNx stores the current cursor X co-ordinate, with 33 being the leftmost position and 2 the rightmost.

$5C89 SPOSNy stores the current cursor Y co-ordinate, with 24 corresponding to the top line, and 3 the bottom line (of the upper screen).

$5C8D ATTR P stores tthe current permanent colour attributes, in the same form as the attribute file.

$5C8E MASK P is used for transparent colours. When any of the bits are set, then the corresponding attribute bit is taken from what is on the scren.

$5C8F ATTR T is similar to ATTR P, but for temporary colours. Most ROM routines use the temporary colours, one notable exception being CLS.

$5C90 MASK T is similar to MASK P, but for temporary attributes.

$5C91 P FLAG stores the current non-colour attributes. The even bits denote the temporary attributes, and the odd ones the permanent. Bits 0 and 1 control OVER, bits 2 and 3 control INVERSE, bits 4 and 5 select INK 9, and bits 6 and 7 select PAPER 9.

$5C92 MEMBOT 30 bytes used as the calculator memory. though also used for creating 'chunky' graphics and as a print buffer when using the STR$ function.

$5CB0 'Not used' will probably be used with the Microdrives and/or RS232, and is referred to in the Non-maskable Interrupt routine (see chapter 7).

APPENDIX F - ROM ROUTINE LIST

This is an alphabetical list of the ROM routines explained in this book. It lists the locations (in hex), which registers are affected by the routines, and the page on which they are explained. The floating point routines not included alter every register.

NAME	HEX	REGISTERS AFFECTED	PAGE
BEEP	03B5	AF,BC,DE,HL,IX	19
BITCALC	22AA	AF,BC,HL	9
BORDER	2297	AF	19
BREAK?	1F54	AF	13
CALCULATE	0E9E	AF,DE,HL	9
CASSETTE	075A	AF,AF',BC,DE,HL,IX	16
CHAR	0018	AF,HL	20
CHUNKY	0B38	AF,BC,DE,HL	8
CIRCLE	232D	All regs inc HL'	11
CLEAR	1EAF	AF,BC,DE,HL	19
CLS	0D6B	AF,BC,DE,HL	10
COLOUR	0BDB	AF,DE,HL	9
COPY	0EAC	AF,BC,DE,HL	15
DRAWcurve	2394	All regs inc HL'	11
DRAWstraight	2477	All regs inc HL'	11
EMPTY	0EDF	AF,BC,DE,HL	14
ERROR	0008	N/A	19
FLOATPT	0028	All regs inc HL'	25
INPUT	15D4	AF,AF',BC',DE'	5
INTERRUPT	0038	NONE	38
LLINE	0EF4	AF,BC,DE,HL	14
LOAD	0556	AF,AF',BC,DE,HL,IX	18
LPRINT	0ECD	AF,BC,DE,HL	14
MAKEROOM	0030	AF,BC,DE,HL	30
NMASKINT	0066	NONE	21
NXTCHAR	0020	AF,HL	20
OPENCHAN	1601	AF,BC,DE,HL	5
OUTPUT	0010	AF,AF',BC',DE'	5
PAUSE	1F3D	AF,BC	19
PLOT	22DF	AF,BC,DE,HL	9

PRINT	0C0A	All regs except HL'	6
PRINTBC	1A1B	AF,AF',BC,BC',DE'	6
PRINTSTRING	203C	All regs except HL'	5
SAVE	04C2	AF,AF',BC,DE,HL,IX	17
SCROLL	0DFE	AF,BC,DE,HL	10
SCROLLsome	0E00	AF,BC,DE,HL	11